DIF Clearinghouse

P.O. Box 638
Newton Lower Falls, MA 02162

DIF TECHNICAL SPECIFICATION

DIF[tm] is the format for data interchange
developed by Software Arts, Inc.

DIF-2B-0882

Limited License to Copy:

This Technical Specification is intended for the use of the original purchaser only. The original purchaser is hereby licensed to copy it for his own use, provided that this notice, together with the copyright, trademark and warranty notices, are reproduced on each such copy. Copying of this document in any form for purposes of resale, license or distribution is prohibited.

No Warranty:

This document is being published to enhance the usefulness of DIF, a format for data interchange, as used by the VisiCalc ® and other programs.

NEITHER SOFTWARE ARTS, INC. NOR THE DIF CLEARINGHOUSE MAKES ANY WARRANTY, EXPRESS OR IMPLIED, WITH RESPECT TO THE QUALITY, ACCURACY OR FREEDOM FROM ERRORS OF THE DIF FORMAT OR OTHER CONTENTS OF THIS DOCUMENT, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR OF FITNESS FOR A PARTICULAR PURPOSE, AND SOFTWARE ARTS, INC. SPECIFICALLY DISCLAIMS ALL LIABILITY FOR DAMAGES RESULTING FROM THE USE OF SUCH FORMAT OR OTHER CONTENTS.

DIF[tm] Technical Specification

1. Introduction

This document is the technical specification of DIF, the data interchange format developed by Software Arts, Inc. It is a reference document and not a tutorial. It includes a description of the DIF file organization and structure, required items, and optional standard items. It also explains the use of the optional standard items by specific applications. Appendix A is an example of a DIF data file.

Programs should use defined standard items when possible. The DIF Clearinghouse will update this document to describe new items as they are defined and record their use in specific programs. Programmers developing new software that incorporates new optional items should inform the Clearinghouse fully about them so that they can be standardized for common use by any program supporting the DIF format.

Programmers should remember that the program reading the data can be extremely simple. The program writing the data must handle it in such a way that it can be read by any program supporting DIF.

Within this text, upper case characters are actual values to be entered as shown and lower case characters name the value to be entered to a field. It is assumed that the ASCII character set is being used. See the section on Definitions for a discussion of character sets.

2. Constraints of the Format

The DIF format was designed for ease of use, and, for the sake of simplicity, certain constraints have been imposed on the format. Because DIF is not intended to be a universal representation for all data, one of these constraints is the representation of data in tables with rows of equal length and columns of equal length. A second constraint is that, because many users program in BASIC, the files must be compatible with BASIC programs. Programs written in another language, such as Pascal, can use a set of subroutines to read and write DIF files.

Below is a list of specific constraints on a DIF file.

1. Because some BASICs have only primitive facilities for reading and writing strings, the convention of keeping numbers and strings on separate lines has been adopted.

2. Two items, VECTORS and TUPLES, are required to support systems that require preallocation of space.

3. Because some systems do not allow programs to test for the end of a file, a special data value, EOD, provides graceful termination to a program.

4. To simplify programming, there are only two formats within the file, and all fields are predefined as character strings or numbers.

5. Strings must be enclosed in quotes if they contain characters other than alphanumerics.

6. The character set is restricted to the printable ASCII characters.

7. Although DIF places no explicit restriction on the length of data strings, some systems may impose restrictions.

Since the DIF format is not meant to meet all the needs for data representation, it may be necessary to use multiple DIF files or additional formats for some applications. A word processor, for example, would not use a DIF file to store text but could use DIF files for tables of values within a report.

3. Organization of the DIF Data File

A DIF file is a text file using the standard printable character set of the host machine. The model for the data is a table. Fields are called vectors; records are called tuples. Data is organized into vectors of equal length. Each tuple consists of a row of corresponding values read across each vector. The user determines the specific groupings of vectors and tuples. Often vectors are treated as columns and tuples are treated as rows, but because DIF can transpose columns and rows, the terms vectors and tuples are used instead of the terms columns and rows.

The DIF file consists of two sections, a header section and a data section. The header section contains descriptions of the file and the data section contains the actual values.

4. The Header Section

The header section is composed of header items. There are four standard required header items and several standard optional header items.

4.1 The Header Item

The header items describe the data organization. Each header item consists of four fields arranged on three lines as illustrated below. The first line is a token[1], the second line consists of two numbers, and the third line contains a string.

```
Topic
Vector Number, Numeric Value
"String Value"
```

4.1.1 The Topic

The first line of the header item is the Topic. It identifies the header item, and must be a token.

4.1.2 The Vector Number

The first field on the second line is the Vector Number. If the header item describes a specific vector, the Vector Number specifies the vector being described. If the header item describes the entire file and not one specific vector, the Vector Number is zero (0).

4.1.3 The Numeric Value

The Value is an integer and occupies the second field of the second line, separated by a comma from the Vector Number. If the header item does not use a numeric value, the Value is zero (0).

4.1.4 The String Value

The String Value occupies the third line of the header item. The String Value is always enclosed in quotation marks. If it is not used, the line consists of a null string, a pair of quotations marks with no space between them.

---

[1] A token is an upper case string of alphanumeric characters. It is usually short, 32 characters or less. See the Definitions section for more information.

4.2 Header Items

There are four required header items. The other header items described in this document are standard optional header items. The defined standard items should be used by new programs using DIF. If it is absolutely necessary, a new header item may be defined to meet the needs of a particular program. For details, see the section on Defining New Header Items.

A program may ignore all header items until it finds the header item DATA, described below.

The following four header items are required:

4.2.1 The First Header Item

```
TABLE
0,version
"title"
```

The header item TABLE must be the first entry in the file. It identifies the file as a DIF file. The version number must be 1. The "title" is the title of the table and describes the data.

4.2.2 Vector Count

```
VECTORS
0,count
""
```

The header item VECTORS specifies the number of vectors in the file.

Note: This header item must appear before header items that refer to vector numbers. Otherwise, it can appear anywhere within the header section.

4.2.3 Tuple Count

```
TUPLES
0,count
""
```

The header item TUPLES specifies the length of each vector (the number of tuples). This can be used by a program to preallocate

storage space for the data. This item may appear anywhere within the header section.

Note: Programs reading the data assume that the tuple count is correct. Some programs may be able to generate this information only after all data has been generated. These programs must reread the DIF file to count the tuples, and rewrite the TUPLES item with the correct count.

4.2.4 The Last Header Item

```
DATA
0,0
""
```

The header item DATA must be the last header item. It tells the program that all remaining data in the file are data values.

The following header items are optional. The programs that are known to use them are noted with the item. For detailed information on each program's specific use of the item, see the section below on Applications Programs.

4.2.5 Vector Label

```
LABEL
vector#,line#
"label"
```

The header item LABEL provides a label for the specified vector. The line number provides an option for labels that span more than one line, and can be ignored by a system that allows single line labels only. The values 0 and 1 are equivalent line numbers.

Note: Some programs do not use the LABEL field. If the first vector in a tuple contains string values, the first data value in the tuple may be treated as a label.

Used by the VisiPlot^tm and VisiTrend/VisiPlot^tm programs.

4.2.6 Vector Comment

```
COMMENT
vector#,line#
"comment"
```

The header item COMMENT is similar to LABEL. It provides an option to systems that allow an expanded description of a vector in addition to a label.

Used by the VisiPlot and VisiTrend/VisiPlot programs.

4.2.7 Field Size

```
SIZE
vector,bytes
""
```

The header item SIZE provides to programs such as data base systems the option to allocate fixed size fields for each value.

Because SIZE is an optional item, programs using SIZE must be able to read files produced by programs unable to generate SIZE information.

Used by the CCA/DMS program.

4.2.8 Periodicity

```
PERIODICITY
vector#,period
""
```

The header item PERIODICITY provides the option of specifying a period in a time series.

Used by the VisiPlot and VisiTrend/VisiPlot programs.

4.2.9 Major Start

```
MAJORSTART
vector#,start
""
```

The header item MAJORSTART specifies the first year of a time series.

Used by the VisiPlot and VisiTrend/VisiPlot programs.

4.2.10 Minor Start

```
MINORSTART
vector#,start
""
```

The header item MINORSTART specifies the first period of a time series.

Used by the VisiPlot and VisiTrend/VisiPlot programs.

4.2.11 True Length

```
TRUELENGTH
vector#,length
""
```

The header item TRUELENGTH specifies the portion of a vector that contains significant values.

Used by the VisiPlot and VisiTrend/VisiPlot programs.

4.2.12 Units

```
UNITS
vector#,0
"name"
```

The header item UNITS specifies the unit of measure for the values in the given vector. Name is the unit, for example meters or ft.

Used by TK!Solver.

4.2.13 Display units

```
DISPLAYUNITS
vector#,0
"Name"
```

The header item DISPLAYUNITS specifies the unit in which the values in the given vector should be displayed. This unit may be different from the one in the UNITS field. The values in the given vector are always stored in the unit specified in the UNITS field, and the application program is responsible for making the

value conversion between the UNITS and DISPLAYUNITS.

For example, a vector might be stored in km, but displayed in the program in miles. The UNITS field would be km, the DISPLAYUNITS field would be miles, and the values in the vector would be in km. Any program using the vector would have to define the conversion between km and miles to display the values in miles.

Used in the TK!Solver program.

4.3 Defining New Header Items

If there is no standard optional header item to fulfill the specific need of a subsystem, a new header item may be defined. Because the DIF format is intended for common use, new optional header items should be standardized through the DIF Clearinghouse. They will then be added to this document.

To be accepted as standard items, new optional items must be consistent with existing conventions.

An optional item extends the format for a specific application. Any program reading the DIF file should be able to operate without optional items. If a reading program requires the information provided by an optional item, it should prompt the user to supply the missing information and not require the item itself.

5. The Data Section

The data section consists of a series of tuples. The Data Values within the tuples are organized in vector sequence.

Each Data Value represents one element of data in the file. The data may be either the actual data or one of the two Special Data Values that mark the beginning of a tuple (BOT) and the end of data (EOD) in the file.

Each Data Value consists of two lines. The first line consists of two fields containing numeric values, and the second line consists of one field containing a string value. The format is:

```
Type Indicator, Number Value
String Value
```

5.1 The Type Indicator Field

The Type Indicator is an integer that tells the program what kind of data is represented by this value. There are currently three possible values.

- -1 The data is a Special Data Value, indicating either the beginning of a tuple or the end of data. The Number Value is zero (0) and the String Value is either BOT or EOD. See the description below of Special Data Values.

- 0 The data is numeric. The Number Value field contains the actual value and the String Value field contains a Value Indicator. See the descriptions below of the Number Value and String Value fields.

- 1 The data is a string value. The Number Value is zero (0) and the String Value field contains the actual string value.

5.2 The Number Value Field

When the Type Indicator is 0, the Number Value field contains the actual value. The value must be a decimal (base 10) number. It may be preceded by a sign (+ or -) and it may have a decimal point. It may be preceded or followed by one or more blanks. If the data value contains an exponent of a power of ten, the value is followed by the letter E and the signed or unsigned exponent power of ten.

Note: This is the only place where DIF allows a non-integer value. Some programs accept only integer values.

5.3 The String Value Field

The contents of the String Value field are dependent on the Type Indicator.

5.3.1 Special Data Value

If the Type Indicator is -1, the String Value is one of the two Special Data Values, BOT or EOD, and the Number Value is 0.

Each tuple begins with the Special Data Value BOT (Beginning of Tuple). If a program cannot generate a VECTORS header item before generating all data, it can use the Special Data Value BOT to determine the number of vectors in the file by counting the number of Data Values between BOTs when it rereads the file. A program can also verify its position in a file by using the BOT Special Data Value.

The Special Data Value EOD (End of Data) indicates the end of data in the file. The EOD occurs at the end of the last tuple in the file. If the program is unable to generate a TUPLES header item before generating all data, it can determine the number of tuples by counting the number of BOTs before the EOD when it rereads the file. A program can also use the EOD Special Data Value to detect the end of the file.

5.3.2 Numeric Value and Value Indicator

If the Type Indicator is 0, the data is numeric, and the String Value is one of the Value Indicators described below. The Value Indicator overrides the value.

A subsystem may define Value Indicators for its own needs. New Value Indicators should be registered with the DIF Clearinghouse.

The Value Indicators currently defined are:

V — Value - This is the String Value most commonly used with a numeric value. The Number Value contains the actual value.

NA — Not Available - The value is marked as not available. The Number Value is 0.

ERROR — The value represents the result of an invalid calculation. The Number Value is 0.

TRUE — Logical value. The Number Value is 1.

FALSE — Logical value. The Number Value is 0.

The String Value can be ignored in favor of the Number Value, or all values with a Value Indicator other than V can be considered nonexistent. Quotes are not permitted around the Value Indicator.

5.3.3 String Value

If the Type Indicator is 1, the String Value is the actual character string. If the value is a token, the quotation marks are optional. However, if there is a beginning quotation mark, there must be a terminating quotation mark.

6. Definitions

This section defines specific characteristics of DIF.

Character Sets

This document assumes use of the ASCII character set. The following characters are permitted:

```
 !"(#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
```

(The first character in this list is a space.)

There are 95 printable characters, including the space. If the host computer has more than 95 characters, the additional characters must be mapped into the 95 ASCII characters to transfer data to another machine.

Some computers permit only 64 characters. When data is transfered to these machines, lower case characters and the characters `{|}~ are mapped into their corresonding upper case characters. If these transformations affect the integrity of the data, associated documentation should specify the effect.

Transfers between character sets should be transparent to most users. To assure compatibility, strings should not contain nonprinting characters.

EBCDIC

EBCDIC is a binary representation of characters and is used primarily for large IBM computers. An awareness of the representation used is not essential, but if files are transferred between machines they must be converted to the standard representation of the host machine.

Because EBCDIC defines more than the 95 standard printable characters, users should avoid the additional characters when preparing data files on an EBCDIC machine.

String Length

Some programs place a length limit on strings that they read. This results in the truncation of long string values. Some systems also limit the length of lines in a data file. Programs should support a minimal string length of 64 characters, but longer ones are preferable.

String Delimiters

Some systems delimit strings with apostrophes instead of quotation marks. When files are

transferred to or from these systems, appropriate changes must be made.

Tokens

A token is a string consisting of upper case alphanumeric characters. It should have a maximum length of 32 characters. Commonly, tokens may or may not be contained within quotation marks; however, a token that is a required string, such as a header item topic, must be represented without quotation marks.

Floating Point Numbers

A floating point number consists of an optional sign and a series of digits followed by an optional decimal point. The number may be followed by the letter E (exponent) and a signed decimal exponent.

Note: Some systems generate the letter D to indicate a double precision floating point number. This is not standard, but it can be read by compatible programs within a single system. When transfering data to other computers, the D must be converted to an E.

7. Applications Programs

This section records the specific use of DIF by applications programs that support it. Programmers who intend to interface with any of these programs should note the specifics listed here. Standardized optional items used by these applications are listed in the general section on Optional Header Items. However, if a program uses a header item that varies significantly from the conventions, it is mentioned only in this section. The accuracy of this information is not guaranteed.

7.1 The CCA/DMS Program

Published and distributed by VisiCorp, Inc.

Uses: SIZE

7.2 The TREND-SPOTTER(R) Program

Published and distributed by Software Resources, Inc.

The TREND-SPOTTER program requires that the DIF file contain either only one tuple or only one vector.

7.3 The VisiCalc(R) Program

Published and distributed by VisiCorp, Inc.

Program created and written by Software Arts, Inc.

The VisiCalc program does not generate the LABEL items. Some programs interfacing to the VisiCalc program have adopted the convention of examining the first Data Value in a tuple, and, if it is a string value, treating it as a label.

7.4 TK!Solver

Published and distributed by Software Arts, Inc.

Uses: UNITS, DISPLAYUNITS

7.5 the VisiPlot and VisiTrend/VisiPlot Programs

Published and distributed by VisiCorp, Inc.

Early versions of the VisiPlot and VisiTrend/VisiPlot programs used the Number Value and String Value incorrectly, storing the Number Value in the String Value field. Programs exchanging data with these versions should check the String Value. If it is not null, the string must be converted and the Number Value computed.

Uses: LABEL, COMMENT, MAJORSTART, MINORSTART, PERIODICITY, TRUELENGTH

I. Sample DIF File

This is an example of a DIF data file. The data in the file is represented by the table below.

PROFIT REPORT

YEAR	SALES	COST	PROFIT
1980	100	90	10
1981	110	101	9
1982	121	110	11

The Data File

```
TABLE                     ------>
0,1                             >
"PROFIT REPORT"                 >
VECTORS   ----> Header          >
0,4             >               >
""        ----> Item            >
TUPLES                          >
0,3                             >
""                              >
LABEL                           >
1,0                             >
"YEAR"                          > Header
LABEL                           >
2,0                             >
"SALES"                         >
LABEL                           >
3,0                             >
"COST"                          >
LABEL                           >
4,0                             >
"PROFIT"                        >
DATA                            >
0,0                             >
""                        ------>
```

```
-1,0                        ---------->
BOT                                   >
0,1980                                >
V                                     >
0,100                                 >
V                                     >
0,90                                  >
V                                     >
0,10                                  >
V                                     >
-1,0                                  >
BOT                                   >

0,1981                ------>         >
V                           >         > Data
0,110                       >         >
V                           >         > Part
0,101 ---> Data             > Tuple   >
V     ---> Value            >         >
0,9                         >         >
V                           >         >
-1,0                        >         >
BOT                   ------>         >
0,1982                                >
V                                     >
0,121                                 >
V                                     >
0,110                                 >
V                                     >
0,11                                  >
V                                     >
-1,0                                  >
EOD                         ---------->
```

II. Sample BASIC program that writes a DIF file

This program enters student records into a file by prompting the user for a student's name and test scores and copying the information into a DIF file.

```
100 REM - THIS PROGRAM CREATES A DIF FILE CONTAINING THE
110 REM - NAME AND TEST SCORES OF A GIVEN NUMBER OF STUDE
120 REM - IT PROMPTS FOR A FILE NAME, THE TOTAL NUMBER OF
130 REM - STUDENTS, AND THE NUMBER OF TEST SCORES FOR
140 REM - EACH STUDENT.  IT THEN PROMPTS FOR A STUDENT'S
150 REM - NAME AND TEST SCORES, AND WRITES THEM TO THE
160 REM - FILE AS A TUPLE.

1000 PRINT "OUTPUT FILE NAME:"; :REM - GET FILE NAME.
1010 INPUT F$
1020 OPEN "O",1,F$              :REM - OPEN FILE FOR OUTP
1030 PRINT "NUMBER OF STUDENTS:";
1035                            :REM - PROMPT FOR NUMBER
1040 INPUT NT                   :REM -    TUPLES.
1050 PRINT "NUMBER OF TEST SCORES PER STUDENT:";
1060 INPUT NV                   :REM - NUMBER OF VECTORS
1070 NV = NV + 1                :REM -    NUMBER OF SCORE
1080 GOSUB 3000                 :REM - USE SUBROUTINE TO
1090                            :REM -    OUTPUT DIF HEAD

2000 FOR I = 1 TO NT            :REM - OUTPUT A TUPLE FOR
2010                            :REM -    EACH STUDENT.
2020    T = -1: V = 0: S$ = "BOT"
2025                            :REM - OUTPUT BOT SPECIAL
2030    GOSUB 4000              :REM -    DATA VALUE.
2040    PRINT "NAME OF STUDENT #";I;
2050    INPUT S$                :REM - GET NAME OF THIS S
2060    T = 1: V = 0            :REM - OUTPUT AS STRING D
2070    GOSUB 4000              :REM -    VALUE.
2080    FOR J = 1 TO NV-1       :REM - PROCESS EACH SCORE
2090       PRINT "SCORE #";J;
2100       INPUT V              :REM - GET SCORE.
2110       T = 0: S$ = "V"      :REM - OUTPUT SCORE AS A
2120       GOSUB 4000           :REM -    VALUE.
2130    NEXT J
2140 NEXT I
2150 T = -1: V = 0: S$ = "EOD"  :REM - OUTPUT EOD SPECIAL
2160 GOSUB 4000                 :REM -    VALUE.
2170 CLOSE 1                    :REM - CLOSE THE OUTPUT F
2180 STOP                       :REM - DONE.
```

```
3000                                  :REM - ROUTINE TO OUTPUT
3010 PRINT#1,"TABLE":PRINT#1,"0,1":GOSUB 3500
3020 PRINT#1,"TUPLES":PRINT#1,"0,";NT:GOSUB 3500
3030 PRINT#1,"VECTORS":PRINT#1,"0,";NV:GOSUB 3500
3040 PRINT#1,"DATA":PRINT#1,"0,0":GOSUB 3500


3050 RETURN
3500                                  :REM - ROUTINE TO OUTPUT
3510                                  :REM -    NULL STRING (""
3520 PRINT#1,CHR$(34);CHR$(34)        :REM - PRINT 2 QUOTATION
3530 RETURN


4000                                  :REM - ROUTINE TO OUTPUT
4010                                  :REM -    VALUE.  T IS TH
4020                                  :REM -    INDICATOR, V IS
4030                                  :REM -    NUMBER VALUE, A
4040                                  :REM -    IS THE STRING V
4050 PRINT#1,T;",";V
4060 PRINT#1,S$
4070 RETURN
5000 END
```

III. Sample BASIC program that reads a DIF file

This program uses the output DIF file from the sample program in Appendix B to calculate an average score and letter grade for each student.

```
100    :REM - THIS PROGRAM READS A DIF FILE CONTAINING THE
110    :REM - TEST SCORES OF A GROUP OF STUDENTS, CALCULAT
120    :REM - AN AVERAGE SCORE FOR EACH STUDENT, MATCHES T
130    :REM - AVERAGE TO A LETTER GRADE, AND PRINTS THE
140    :REM - STUDENT'S NAME, AVERAGE, AND LETTER GRADE.


500 DIM T(100)              :REM - MAXIMUM OF 100 VECTORS.
510 DIM V(100)              :REM - T IS THE TYPE INDICATOR, V
520 DIM V$ (100)            :REM -    THE NUMBER VALUE, AND V
530                         :REM -    THE STRING VALUE OF EAC
535                         :REM -    VALUE.
540 GOSUB 5000              :REM - INITIALIZATION SUBROUTINE.
550 GOSUB 6000              :REM - SUBROUTINE TO READ HEADER.
560 FOR I = 1 TO NT         :REM - FOR EACH TUPLE,
570    GOSUB 7000           :REM -    GET ALL VECTOR ELEMENTS
575                         :REM -    TUPLE.
580    M=0                  :REM -    M IS THE SUM OF THE SCO
590    FOR J = 1 TO NV      :REM -    FOR EACH VECTOR VALUE,
600       IF T(J)=1 THEN PRINT V$(J)   :REM - PRINT NAME.
610       IF T(J)=0 THEN M = M+V(J)    :REM - ADD SCORES.
620    NEXT J
630   M = M/(NV-1): PRINT M :REM - PRINT STUDENT'S AVERAG
640   IF M<=50 THEN PRINT "THIS STUDENT'S FINAL GRADE IS
650   IF M<=70 AND M>50 THEN PRINT "THIS STUDENT'S FINAL
660   IF M<=85 AND M>70 THEN PRINT "THIS STUDENT'S FINAL
670   IF M<=94 AND M>85 THEN PRINT "THIS STUDENT'S FINAL
680   IF M>94 THEN PRINT "THIS STUDENT'S FINAL GRADE IS A
690 NEXT I
700 CLOSE 2
710 PRINT "FINISHED CALCULATING GRADES"
720 STOP


5000                        :REM - INITIALIZATION CODE.
5010 PRINT "FILE NAME";
5020 INPUT F$
5030 OPEN "I",2,F$          :REM - OPEN FILE FOR INPUT.
5040 NV = 0                 :REM - INITIAL VECTOR COUNT.
5050 NT = 0                 :REM - INITIAL TUPLE COUNT.
5060 RETURN
```

```
6000   :REM - READ HEADER. GET NUMBER OF VECTORS AND TUPL
6010 INPUT#2,T$           :REM - GET TOPIC.
6020 INPUT#2,S,N          :REM - GET VECTOR NUMBER AND VALU
6030 INPUT#2,S$           :REM - GET STRING VALUE.
6040 IF T$="VECTORS" THEN 6500:REM - CHECK FOR KNOWN HEAD
6050 IF T$="TUPLES" THEN 6600 :REM -     ITEMS.
6060 IF T$="DATA" THEN RETURN
6065                      :REM - "DATA" ENDS HEADER.
6070 GOTO 6010            :REM - IGNORE UNKNOWN ITEMS.
6500 NV = N               :REM - NUMBER OF VECTORS.
6510 IF NV<=100 THEN 6010 :REM - CHECK FOR 100 OR LESS VE
6520 PRINT "TOO MANY VECTORS. PROGRAM CAPACITY 100 VECTOR
6530 CLOSE 2
6540 STOP
6600 NT = N               :REM - NUMBER OF TUPLES.
6610 GOTO 6010            :REM - GET NEXT HEADER ITEM.


7000   :REM - SUBROUTINE TO GET ALL VECTOR ELEMENTS IN A
7010 GOSUB 8000           :REM - GET NEXT DATA VALUE.
7020 IF T1<>-1 THEN 9000 :REM - MUST BE BOT, ELSE ERROR
7030 IF S$<>"BOT" THEN 9000
7040   FOR K = 1 TO NV :REM - GET EACH DATA VALUE.
7050      GOSUB 8000
7060      IF T1>1 THEN 9000
7070      T(K) = T1       :REM - SAVE TYPE INDICATOR.
7080      V(K) = V1       :REM - SAVE NUMBER VALUE.
7090      V$(K) = S$      :REM - SAVE STRING VALUE.
7100      NEXT K
7110 RETURN


8000    :REM - SUBROUTINE TO GET NEXT DATA VALUE.
8010 INPUT#2,T1,V1        :REM - GET TYPE INDICATOR, NUMERIC
8020 INPUT#2,S$           :REM -    VALUE, AND STRING VALUE.
8030 RETURN

9000    :REM - ERROR ROUTINE
9010 PRINT "ERROR IN FILE FORMAT"
9020 CLOSE 2              :REM - END PROGRAM
9030 STOP
9040 END
```

IV. Sample Pascal program using a DIF file

This program is a Pascal program that reads data from a DIF file into an array and displays the results on the terminal.

```
{ This is a simple program which reads DIF file data into an arra
  displays the results on the terminal.  It makes use of a proced
  called "get_dif_array" which handles only numeric data.  It is
  for Apple Pascal 1.1 and may require modification to run on oth

program dif_read;

const
   max_vector =  10;                                  { maximum number of
   max_tuple  =  10;                                  { maximum number of

type
   vector_index = 0..max_vector;
   tuple_index  = 0..max_tuple;
   dif_array    = array[1..max_vector, 1..max_tuple] of real;

var
   in_file      : text;            num_vectors : vector_index;
   fname        : string[15];      num_tuples  : tuple_index;
   matrix       : dif_array;       code, i, j  : integer;
```

```
{ "Get_dif_array" reads a DIF file and returns the file data (cur
  only numeric) in an array.  Also returns number of vectors and
  -- these must be specified in file header -- and an error code.

procedure get_dif_array (var dif_file: text; var real_array: dif_
                         var nvectors: vector_index; var ntuples:
                         var return_code: integer);

   const                          { currently defined data types }
       special = -1; numeric =  0; char_string  =  1; other    =

   type
       header_item  = record
                        topic          : string;
                        vector_num     : vector_index;
                        value          : integer;
                        string_value : string
                      end;
       data_value   = record
                        kind           : -1..2; { currently defined
                        number_value : real;
                        string_value : string
                      end;

   var
       hdr_item       : header_item;
       data_val       : data_value;
       tuple, vector : integer;


   { "Read_integer" reads an integer terminated with a comma.  Th
     is required because this Pascal dialect's "read" procedure r
     only <space>, eoln and eof as delimiters of integer values.

   procedure read_integer (var number: integer);
      var
         sign, magnitude : integer;
         ch : char;
      begin
         sign := 1;  magnitude := 0;                  { initialize }
         read (difile, ch);                           { get 1st chara
         while ch <> ',' do                           { comma is deli
            begin
               case ch of
                 '-' : sign := -1;
                 '0','1','2','3','4','5','6','7','8','9'
                     : magnitude := magnitude * 10 + ord(ch) - or
               end;  { case }
               read (difile, ch)                      { get next char
            end;
         number := sign * magnitude                   { return result
      end;  { read_integer }
```

```
{ "Read_string"  deletes leading and trailing blanks and strip
  quotes from quoted strings. }

procedure read_string (var str: string);
   begin
      readln (difile, str);
      while str[1] = ' ' do                        { leading blank
         delete (str, 1, 1);
      if str[1] = '"'                              { strip quotes
         then begin
                delete (str, 1, 1);
                delete (str, pos('"', str),
                        length(str) - pos('"', str) + 1)
              end
      else if pos(' ', str) > 0                     { trailing bla
         then delete (str, pos(' ', str),
                         length(str) - pos(' ', str) + 1)
   end;  { read_string }
```

```
procedure read_header_item (var item: header_item);
   begin
      read_string (item.topic);                       { get topic }
      read_integer (item.vector_num);                 { get vector n
      readln (difile, item.value);                    { get value }
      read_string (item.string_value)                 { get string v
   end;    { read_header_item }

procedure read_data_value (var value: data_value);
   begin
      read_integer (value.kind);                      { get data typ
      readln (difile, value.number_value);            { get number v
      read_string (value.string_value)                { get string v
   end;    { read_data_value }

begin   { get_dif_array }
   return_code := 0;                                  { assume no pr
   nvectors := 0; ntuples := 0;                       { initialize }
   repeat                                             { read header
      read_header_item (hdr_item);
      if hdr_item.topic = 'VECTORS'
         then nvectors := hdr_item.value              { vector count
      else if hdr_item.topic = 'TUPLES'
         then ntuples := hdr_item.value               { tuple count
   until hdr_item.topic = 'DATA';

   if (nvectors = 0) or (ntuples = 0)                 { check counts
      then return_code := 1
      else begin
           for tuple := 1 to ntuples do               { read data }
             begin
               read_data_value (data_val);                 { BOT }
               for vector := 1 to nvectors do
                 begin
                   read_data_value (data_val);
                   if data_val.kind = numeric
                     then real_array[vector, tuple] :=
                                    data_val.number_value
                 end
             end;
           read_data_value (data_val);                     { EOD }
           if (data_val.kind <> special) or
              (data_val.string_value <> 'EOD')
              then return_code := 2
           end

end;  { get_dif_array }
```

```
begin    { dif_read }

   writeln;                                    { get DIF file name }
   write ('DIF file name: ');
   readln (fname);
   reset (in_file, fname);                     { open and point to BOF }

   get_dif_array (in_file, matrix, num_vectors, num_tuples, code)

   close (in_file);                            { close DIF file }
   case code of                                { display results }
     0: begin
          writeln;
          writeln ('"', fname:15, '"', ' contains ', num_vectors:
                   ' vectors and ', num_tuples:3, ' tuples.');
          writeln ('The data values follow in tuple order:');
          writeln;
          for i := 1 to num_tuples do
            begin
              for j := 1 to num_vectors do
                 write (matrix[j, i]:10:2);
              writeln
            end;
          writeln
        end;
     1: writeln ('Error. Tuple or vector count not found.');
     2: writeln ('Error. Data not properly terminated.')
     end   { case }

end.  { dif_read }
```